Attorney Docket No.: 002435.P002 Patent Application

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

TITLE OF THE INVENTION

NETWORK TRANSACTION PORTAL TO CONTROL MULTI-SERVICE PROVIDER TRANSACTIONS

INVENTOR

LAKSHMI ARUNACHALAM

Prepared by

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026
(303) 740-1980

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number: EL 807366935 US
Date of Deposit: May 23, 2001
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Commissioner of Patents and Trademarks, Washington, D. C. 20231

Krista Mathieson
(Typed or printed name of person mailing paper or fee)

Krista Mathieson
(Signature of person mailing paper or fee)

May 23, 2001
(Date signed)

Network Transaction Portal To Control Multi-Service Provider Transactions

COPYRIGHT NOTICE

[0001] Contained herein is material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction of the patent disclosure by any person as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all rights to the copyright whatsoever.

[0002] This application claims priority to U.S. Provisional Application No. 60/206,422 filed on May 23, 2000 and to application Serial No. 09/792,323, filed on February 23, 2001, which is a divisional patent application of Application Serial No. 09/296,207, filed April 21, 1999, which is a continuation-in-part patent application claiming priority to U.S. Pat. No. 5,987,500, entitled *Value-Added Network System For Enabling Real-Time, Bi-Directional Transactions On A Network*, formerly Application Serial No. 08/879,958, and filed on June 20, 1997. U.S. Pat. No. 5,987,500 is itself a divisional application claiming priority to U.S. Pat. No. 5,778,178, entitled *Method And Apparatus For Enabling Real-Time Bi-Directional Transactions On A Network*, formerly Application Serial No. 08/700,726, filed on August 5, 1996, which also claims priority and benefit under 35 U.S.C. 119(e) to U.S. Provisional Application No. 60/006,634 filed on November 13, 1995.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003] The invention relates generally to performing transactions on a network. More particularly, the invention relates to a system and method for controlling a transaction involving multiple service providers.

Background Information

[0004] The Internet and the World Wide Web, hereinafter referred to as the web, provide a viable medium for electronic commerce and on-line services, however current systems and methods for using the Internet and the Web are extremely limited. In particular, current uses are limited to either browse-only interactions or simple "deferred" purchases involving a single service provider.

[0005] Figure 1 conceptually illustrates a prior art use 100 of the Internet and the web. A user 105 accesses a car dealer web server 155 associated with a car dealer 150 over the Internet 130 via a web browser 110. Web browser 110 is software that runs on a computer system and provides a simple user interface to allow access to web servers via the web. In particular, the user 105 may input a uniform resource locator (URL), such as http://www.cars.com, which the web browser 110 communicates to the Internet 130 and which corresponds to an IP address 120 that uniquely locates the car dealer web server 155 and a web page 160. The user 105 may view the web page 160 and then leave, which amounts to a simple browse-only interaction.

[0006] Alternatively, the user 105 may make a limited, deferred purchase of a car from the car dealer 150 and involving only the car dealer 150. For example, the user 105 may fill out a form on car dealer web page 160 and email the form to car dealer web server 155. After receiving the form, the car dealer web server 155 may perform some processing of the form, and then send it through a gateway 170 towards applications 175 that perform further purchase processing and read and write data 180 such as to a legacy database. The applications 175 and the data 180 are not directly connected to the Internet or the web and are not available to other entities connected to the Internet. Typically, the car dealer 150 alone may access the applications 175 and the data 180, and typically this is via a complicated and customized procedure. The actual purchase is deferred until the email is received, read by a person or system, and purchase processing is performed by a person or the applications 175 and data 180. Thus, the purchase is not performed in real-time and involves only the car dealer 150.

[0007] The user 105 may also select a bank hyperlink 165 embedded in web page 160. The bank hyperlink 165 causes the web browser 110 to connect to bank web server 192 presenting bank web page 194 via hyperlink address 165. This may allow the user 105 to browse bank web page 192 to obtain information about obtaining a loan, however, the association between the car dealer 150 and the bank 190 is a limited one involving the car dealer 150 only providing easy access to bank information via the bank hyperlink 165. Unfortunately, there is no cooperation or interaction between the car dealer 150 and the bank 190 besides the hyperlink 165. In fact, the hyperlink 165 disconnects the user from car dealer web server 155 and web page 160 and connects the user with bank web server 192 and bank web page 194. This lack of cooperation, control, and interaction greatly limits the services that may be provided by the web.

[0008] Figure 2 conceptually illustrates a user 205 and a bank web server 250 interacting dynamically through the use of Common Gateway Interface (CGI) applications. The user 205 accesses the bank web server 250 via a web browser 210 to attempt to obtain information on a checking account and a loan account. The bank web server 250 includes a CGI interface 252 to a checking application 254 and a CGI interface 256 to a loan application 258 that interact with checking data 272 and loan data, respectively, in a database 270. CGI allows the bank web server 250 to transfer data to the checking application 254 and the loan application 258 that can then perform processing on the data. By way of example, the user 205 may enter a checking account identification number in an HTML form provided by the bank web server 250, and the server 250 may communicate the checking account identification number to checking application 254 that uses CGI to look up the user checking account in the database 270 and format the checking account data 272 as an HTML page that may be presented to the user 205.

[0009] However, the CGI interaction is severely limited because each CGI application must be customized for a particular type of application or service. That is, different CGI application would have to be created for each service provided by the

bank. For this reason, creating and managing individual CGI scripts for each service is not a viable solution for merchants with a large number of services.

[0010] As the Web expands and electronic commerce becomes more desirable, the need increases for robust, real-time, bi-directional transactional capabilities on the Web. A true real-time, bi-directional transaction would allow a user to connect to a variety of services on the web, and perform real-time transactions on those services. For example, although user 100 can browse car dealer Web page 105 today, the user cannot purchase the car, negotiate a car loan or perform other types of real-time, two-way transactions that he can perform with a live salesperson at the car dealership.

[0011] Ideally, user 100 in FIG. 1A would be able to access car dealer Web page 105, select specific transactions that he desires to perform, such as purchase a car, and perform the purchase in real-time, with two-way interaction capabilities. CGI applications provide user 100 with a limited ability for two-way interaction with car dealer Web page 105, but due to the lack of interaction and management between the car dealer and the bank, he will not be able to obtain a loan and complete the purchase of the car via a CGI application. The ability to complete robust real-time, two-way transactions is thus not truly available on the web today.

[0012] In order to provide sophisticated and useful services over the web, it is desirable to control and manage cooperation and interaction among a plurality of service providers that each contribute to the transaction. This goal is constrained by the prior art systems and methods for using the Internet, which do not control or manage multi-service provider transactions and which do not permit sophisticated and useful joint service offerings.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

[0013] The novel features believed characteristic of the invention are set forth in the appended claims. The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements. The invention itself, however, as well as a preferred mode of use, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings:

[0014] **Figure 1** conceptually illustrates prior art uses of the Internet.

[0015] **Figure 2** conceptually illustrates prior art uses of CGI applications to provide a dynamic interaction between a user and a web server.

[0016] **Figure 3** conceptually illustrates a system that includes service network processing to allow a transaction involving multiple service providers, according to one embodiment of the invention.

[0017] **Figure 4** conceptually illustrates relationships between components of a service network system, according to one embodiment.

[0018] **Figure 5** conceptually illustrates a service network that allows controlled, sophisticated, interactive, "any-to-any", real-time, services to be provided by multiple service providers, according to one embodiment.

[0019] **Figure 6** conceptually illustrates a hub-controlled service network 600, according to one embodiment.

[0020] **Figure 7** conceptually illustrates a service network system showing a hub creating controlled links to multiple nodes, according to one embodiment.

[0021] **Figure 8** conceptually illustrates a service network system with an application environment that is connected to the service network.

[0022] **Figure 9** conceptually illustrates in block diagram form a method, according to one embodiment, to perform a transaction on a service network.

[0023] **Figure 10** conceptually illustrates components of a service network, according to one embodiment.

[0024] **Figure 11** conceptually illustrates a hierarchical branching convention to provide network addresses for networked objects, according to one embodiment.

[0025] **Figure 12** conceptually illustrates a hub-controlled service network providing verified services, according to one embodiment.

[0026] **Figure 13** conceptually illustrates the Open System Interconnection (OSI) reference model.

[0027] **Figure 14** conceptually illustrates a layered architecture of a transactional network application having a value-added network (VAN) switch, according to one embodiment.

[0028] **Figure 15** conceptually illustrates the potentially distributed nature of a VAN switch, according to one embodiment.

[0029] **Figure 16** conceptually illustrates software layers of an object router, according to one embodiment.

[0030] **Figure 17** conceptually illustrates data model integration for an object router of one embodiment.

[0031] **Figure 18** conceptually illustrates a single bank service transaction, according to one embodiment.

[0032] **Figure 19** conceptually illustrates a multi-service provider transaction, according to one embodiment.

[0033] **Figure 20** conceptually illustrates an exemplary architecture for a bank transaction, according to one embodiment.

[0034] **Figures 21-22** conceptually illustrate an exemplary class diagram showing object classes to implement one embodiment.

[0035] **Figure 23** conceptually illustrates a timing diagram for a router, according to one embodiment.

[0036] **Figure 24** conceptually illustrates a Finite State Machine (FSM), according to one embodiment.

[0037] **Figure 25** conceptually illustrates an Extended Finite State Machine (EFSM) counter, according to one embodiment.

[0038] **Figure 26** conceptually illustrates code processing, according to one embodiment.

[0039] **Figure 27** conceptually illustrates code of a CoreBusinessObject, according to one embodiment.

[0040] **Figure 28** conceptually illustrates an exemplary Distributed Online Service Information Base (DOLSIB) FSM diagram for a bank, according to one embodiment.

[0041] **Figure 29** conceptually illustrates a diagram with expect, found, and error states.

[0042] **Figure 30** conceptually illustrates an exemplary Distributed Online Service Information Base (DOLSIB) FSM diagram for another bank, according to one embodiment.

[0043] **Figure 31** conceptually illustrates operation of a hub and node service control system, according to one embodiment.

[0044] **Figure 32** conceptually illustrates an architecture, according to one embodiment, to manage a hub and node system.

[0045] **Figure 33** is a block diagram of a computer system upon which one embodiment may be implemented.

DETAILED DESCRIPTION OF THE INVENTION

[0046] A method and apparatus are described for performing transactions involving multiple service providers over a service network. Broadly stated, embodiments of the present invention seek to maintain control over the transaction including controllably and selectively routing to and involving service providers in the transaction. According to one embodiment, this may include a network transactional application including control and routing software objects and distributed remote software objects to interface with the network transactional application and perform controlled transactions. Advantageously, this may allow sophisticated, real-time, multi-service provider transactions to be performed while allowing one entity (e.g., a context owner) to control the transaction.

[0047] In the following description, for the purpose of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form.

[0048] Figure 3 conceptually illustrates a system 300, according to one embodiment, that includes service network processing that allows a controlled transaction involving a plurality of networked service providers to be performed. A client access device 310 connects to, accesses, or otherwise communicates with a facilities network 320 that contains service network processing 350. The term "client access device" will be used to broadly refer to a device to access the facilities network and may be a computer system, a computer system with a web browser, a personal digital assistant, a mobile end point, a cellular device (e.g., a cell phone), a screen phone, a pager, a home appliance (e.g., a TV, VCR, etc.), a remote control device to a TV or VCR, an ATM machine, a cash register, and other devices.

[0049] The client access device 310 may access the facilities network and the service network via server "switching" sites or corresponding appropriate non-web switching sites such as cellular provider sites. For example, a cell phone may access a

cell site where resides a computer system having an IP address and a functional connection to a hub either on that computer system or on a connected computer system.

[0050] The facilities network may be any suitable facilities network or combination of potentially heterogeneous facilities networks, including an IP-based network, a TCP/IP-based network, the Internet, the web, a non-web network, an email network, Integrated Services Digital Network (ISDN), Asynchronous Transfer Mode (ATM), Personal Communications Services (PCS), X.25, Ethernet, frame relay, token ring, Fiber Distributed Data Interface (FDDI), Community Antenna TV (CATV), an intelligent network, a public-switched network, a public-switched telephone network, a plain old telephone system (POTS) network, a private switched network, a wireless network, a cellular network, private/leased lines, an intranet, a private enterprise network, or another network suitable for supporting a service network such as those described in the present specification. For example, the client access device 310 may connect to the facilities network via a wire, cable, cellular, or PCS connection, service provider 1 360 may connect via a T1 connection, service provider N 380 may connect via a T3 connection, and service provider N 380 may additionally have an ATM/Sonet or Frame Relay/T3 connection to a branch office to perform processing.

[0051] The service network processing 350 is functionally interposed between the client access device 310 and multiple service providers and associated software that provides services to the client access device 310. According to one embodiment, the service network processing 350 provides a network transactional application that provides an overlay service network that operates on and runs on the facilities network 320. The network transactional application may provide the service network according to an N-tier manager-agent model that achieves N-way communication by using a value-added network (VAN) switch or object router that resides at the transaction network entry point to route to software residing at remote service provider nodes. The network application may use an N-way interactive object router to provide the link between the clients and the service providers. The service network may provide access to a myriad of network services such as selling of products (e.g.,

books) and services (e.g., shipping, pizzas delivery), banking, trading (e.g., stocks), advertising, customer service, bill management, and others.

[0052] The service network processing 350 may include transactional control and management software to control and manage one or more transactions involving the client access device 310 and any number of service providers that are desired for the particular transaction. Control and management may include establishing a connection or link (e.g., a line, channel, or thread over which data may be communicated) to service providers, making requests, activating or configuring transactional applications, receiving results, de-establishing connections with service providers, fault-handling, monitoring performance, monitoring transactions, monitoring client activity and service provider activity (e.g., to support accounting and billing policies of the service network), collecting statistics, security processing, address processing to uniquely address and identify network locations and objects by a unique network address, routing processing to uniquely identify, retrieve, and route dynamically changing information and software objects using multi-media, object routing, and others. According to one embodiment, management includes distributed control of Events, Configuration, Accounting, Performance, and Security (ECAPS). By way of example, events may include responding to specific occurrences on the network, configuration may include managing the connections that exist within the network, accounting may include measuring and recording network transaction activities, performance may include monitoring and maintaining network performance standards, and security may include enabling connection and transaction privacy.

[0053] The service network processing 350 may support industry-standard web browsers (e.g., Internet Explorer available from Microsoft Corporation of Redmond Washington), web servers, security protocols, and connect to applications and middleware, including both legacy and relational database management systems (RDBMSs). In an embodiment where the service network 350 operates over the Internet, the service network 350 may comply with open Internet standards and protocols.

[0054] According to various embodiments, the network processing 350 may be distributed between a hub and a plurality of nodes each associated with a service provider. The term "hub" will broadly be used to refer to one or more functionally coupled computer systems (e.g., a web server server) that provide software and methods to control a transaction or service involving multiple service providers. The hub may be considered as a portal or gateway into the service network that provides selective and controlled access into the service network to computer systems and methods associated with service providers of the network. The term "node" will broadly be used to refer to one or more functionally coupled computer systems that provide service methods under the control of the hub. Thus, the service network 350 may extend to software, objects, and methods at the service providers 360, 370, and 380, as will be explained in more detail elsewhere. According to one embodiment, links from the client access device 310 to such software, objects, and methods is via the hub.

[0055] According to certain embodiments the hub includes a router to route to and establish links to software objects at nodes. The term "router" will broadly be used to refer to software to create or allow a link to potentially remote and geographically distributed software. In one embodiment, the router is an object router that uses objects and class information rather than unrelated functions and data. For example, in one specific embodiment, the object router provides two types of a class, namely a skeleton that is the functionality of the object and its stub that allows remote access to the same object. Typically the stub is located on one computer system, such as a client computer system or a hub computer system and the skeleton is located on a different remote and geographically distributed computer system associated with a service provider. a user to specify functions to be executed remotely. According to one embodiment, the object router is part of a value-added network transactional application and resides at the network entry point (e.g., the hub) to provide an N-way interactive link to other software that resides at remote and geographically distributed IP nodes. Specific exemplary embodiments of hubs, nodes, routers, object routers,

and objects will be discussed elsewhere, although other embodiments are contemplated.

[0056] The term "service provider" will be broadly used to refer to a network-connected entity or presence, such as a business, merchant, organization, administration, networked user, or other provider that provides or participates in a service associated with the service network. Typically, the service provider participates in joint services involving multiple service providers. The multiple service providers may include a service provider 1 360, a service provider N 380, and optionally any number of additional service providers 370. Accordingly, the total of service providers may be any integer number of service providers. The service providers 360, 380 may be any service providers suitable for the intended service network, including merchants and businesses that desire to provide their products and/or services to a client associated with the client access device 310. For example, the service providers may be businesses that provide web servers, web pages, transactional applications to sell products or services, and data to facilitate the transaction. The multiple service providers may also include other client access devices similar to client access device 310. For example, client access device 310 may obtain services that involve other client access devices, such as in a service network incorporating features similar to those in an interactive chat or messaging, an online bartering, an online file-sharing, or other services. The service providers are to be interpreted broadly in the present application and many exemplary service providers will be discussed in the specification, although others are contemplated.

[0057] Figure 4 conceptually illustrates a system 400, according to one embodiment, to provide services via a service network. A client 405 uses a client access device 410 such as a web browser 411, a cell phone 412, a television 413 (e.g., web-enabled television, and others), or another client access device 414, such as a kiosk or an ATM machine, to access a facilities network 420. The facilities network may include a carrier network 422 such as one or more of a telco, wireless, CATV, or other carrier network. This may include cables, radio frequency, satellite, fiber optic, and other links. Alternatively, a client or user may walk-in 424 to client access

devices such as the kiosk or ATM machine, which may be at a bank, a store, a mall, or another public place. In the case of a web browser access device, connecting to the facilities network may include connecting to an Internet service provider 426 to obtain access to a web server 428 offering a web page. In the case of a cell phone access device 412, connecting to the facilities network may include a dial-up connection 430. In the case of a television access device 413, accessing the network may include using buttons on the television or on a remote control along with optional list or menu options to connect to the facilities network 432. In the case of a kiosk or ATM device, a client or user may interact with the kiosk or ATM device, that may either connect to the facilities network or already be connected to the facilities network.

[0058] After accessing the facilities network, the client access device 410 may access or utilize the service network 435. This may be done in a way compatible with the client access device 410 and the service network 435. For example, a web browser access device 411 may request to access the service network to obtain one or more services by communicating a request to connect to a web server and web page based on data input into a computer system by a client or user. Alternatively, a cell phone access device 412 may access the service network by entering a phone number associated with the service network 435 into the cell phone access device 412, which automatically connects to the facilities network 420 and the service network 435, which may be a call center providing interactive voice response (IVR).

[0059] The service network 435 may selectively and controllably manage the connection to and use of service provider hardware and software 440, which may be by direct connection 450 or by indirect connection 460 with the service network 435. As shown, applications 451, middleware 452, 4GL applications 453, operating systems 454, and hardware 455 may be directly connected to the service network 450. Typically data 462 (e.g., enterprise data), host TP applications 464, and other hardware 466 (e.g., printers, faxes, etc.) will be indirectly connected to the service network 460, such as via the applications 451, or middleware 452.

[0060] Figure 5 conceptually illustrates a service network 500 that allows controlled, sophisticated, interactive, "any-to-any", real-time, services to be provided

by multiple service providers. A client access device 510 connects with the service network and then receives a service involving cooperation between a service provider #1 520, a service provider #2 530, and optionally any desired number of additional service providers 540. The client access device 510 may bi-directionally communicate and interact with the service provider #1 520 by link 550. Likewise, the access device 510 may interact with the service provider #2 via link 555. As shown by link 560, the service provider #1 and service provider #2 may also interact directly, rather than via the client access device 510. Links 565, 570, 575 may also be provided when one or more other service providers 540 are desired.

[0061] Without loss of generality to other services and transactions, and to illustrate the advantages provided by the service network, consider an exemplary multi-provider service involving a client access device 510 purchasing a car from car dealer presence 520 by obtaining a loan for the amount of the car from bank presence 530 and insurance for the car from insurance provider 540. The client access device 510 first determines the amount of the car from car dealer presence 520 and indicates a desire to purchase the car for the amount by obtaining a loan from bank presence 530.

[0062] Then, the service network automatically establishes a controlled link 560 with bank presence 530. Advantageously, this may be done without losing connection to and communication with car dealer presence 520. Then, bank presence 530 establishes a controlled link 555 with access device 510 to obtain data to process the loan. After bank presence 530 approves the loan it may verify the loan to the client via controlled link 555 and to the car dealer presence 520 via controlled link 560.

[0063] The car dealer presence 520 may then connect with an insurance provider 540 via controlled link 565 to advertise an insurance policy to client access device 510 via controlled link 575 and receive an acceptance of the policy via controlled link 575. The insurance provider 540 after processing the insurance policy may provide verification to the client via controlled link 575 and to the car dealer presence via controlled link 565. The car dealer presence 520 may then send a

complete transaction verification and summary to the client access device 510 via controlled link 550 to finalize the particular transaction.

[0064] As discussed, the transaction involves interactions between the client access device 510 and multiple service providers 520, 530, 540 under the control of the service network. Advantageously, this allows collaborative and cooperative transactions and interactions that are not possible in prior art approaches. In this way, transactions are not limited to two-way transactions involving a client access device and a single service provider, but are flexible to include three-way, four-way, or N-way transactions and interactions involving any desired number of service providers and the client access device. According to one aspect of the present invention, predetermined strategies involving collaboration and cooperation among service providers may be devised to incorporate N service providers, where N is any desired integer number of service providers that have agreed to collaborate and cooperate to provide the services. Thus, according to one embodiment, the service network may allow for service transactions involving "any-to-any" communication and interaction, thus facilitating a large, flexible variety of robust, real-time transactions on the network.

[0065] Figure 6 shows a hub-controlled service network 600, according to one embodiment. A client access device 610 accesses a service network via a network entry point 620. The network entry point 620 will typically correspond to and be compatible with the client access device 610. Thus, depending on the access device 610, the network entry point 620 may be provided by the web, a web page, a hyperlink, an application, a call center, a cell site, a TV Head-End Station, or others. For example, for a web-based access device, the network entry point 620 may be provided by a web page (e.g., a web page hyperlink), an application running on the client access device (e.g., a Java Applet running in a web browser), while for a cellular access device, the network entry point 620 may be provided by a phone number to a call center.

[0066] The network entry point 620 allows connection with a hub 630. The hub 630 may serve as a service network control center or network operator to

configure, provision, control and manage access to and services provided by multiple potentially geographically distributed service nodes that provide networked services to clients or subscribers. Advantageously, this allows control and customization of the class and level of service provided over the network by the service control nodes.

[0067] Typically, the hub 630 includes software to control and manage transactions over the service network. According to one embodiment, the hub may assist with providing the network entry point 620 and access to point-of-service applications by providing software such as Java applets or ActiveX controls. The hub 630 may also include multi-protocol value-added network switching software to switch between remote service provider nodes and routing software to perform controlled routing electronic transactional documents, components, objects, or data, in a form that may be received and interpreted by computer systems, applications, hardware, and other networked components associated with the service providers. The hub 630 may also track and store data such as transaction statistics.

[0068] The hub 630 may access a plurality of nodes 640. As shown, the plurality of nodes include a node 1 650, a node 2, optionally any desired number of additional nodes 670, and a node N 680. The node N 680 may represent any desired number of nodes. Typically, each node will be associated with at least one service provider. In one case, a service provider may provide its services through a logical plurality of nodes based on access device, service or product offering, other service providers, and for other reasons. By way of example, a service provider may provide one node for web-based access, one node for cellular access, one node for each major service or product line, one node for business partners, one node for employees, and for other reasons. Additionally, multiple service providers may share a common node. For example, the car dealer and the bank may decide to share a common node.

[0069] Each node may serve as a gateway, portal, or entry point into a private or enterprise network of the service provider. The node may provide selective access to service related resources of the service provider such as applications, data, hardware, personal, and other resources. The node may act as a service agent and

management station for the service provider. It may also provide a channel interface to back-office transaction processing applications.

[0070] According to one embodiment, the hub 630 and the nodes 640 contain software to control and manage a plurality of distributed service and application software objects or components. The term "objects" will be used to refer to separable software objects capable of being distributed over a network and operated remotely. The objects may be object-oriented software objects based on object class. They may be objects conforming to standards and models, such as the Component Object Model (COM), Object Linking and Embedding (OLE), ActiveX, Distributed COM (DCOM), System Object Model (SOM), Distributed SOM (DSOM), Common Object Request Broker Architecture (CORBA), Distributed interNet Applications Architecture (DNA), COM+, Java-based components, and others. For purposes of illustration, and without limitation, a COM object may have a "published" unvarying interface that exposes its service or business functionalities and the parameters it accepts, and the COM object may be accessed in a distributed computing environment by a COM-compliant service application to use its functionalities to deliver services or transactions to a client. Thus, the hub 630 and/or the nodes 640 may provide "component-oriented middleware" that controls and manages potentially distributed components to create distributed applications and provide the service network. The middleware may include management instructions to use the components to deal with transactions, component packaging, and state management. Typically, the hub 630 will contain software to intelligently switch to, route to, configure, provision, track, manage, and control the objects or components. Such an architecture may be well suited to a high throughput transactional environment.

[0071] According to one embodiment, the node uses an intelligent state management engine such as a Distributed Online Service Information Base (DOLSIB) to store and access transaction management information. DOLSIBs will be described in more detail elsewhere in the specification. The node may use the intelligent state management engine or DOLSIB to automatically create the associations between the

clients screen elements and the service objects routed to the service control nodes. Each node may have a separate DOLSIB, according to one embodiment.

[0072] According to one embodiment, software for the hubs and nodes may be provided as shrink-wrapped software packages. The context owners and service providers may then obtain these software packages, input business and management objects into the DOLSIB, and create or join service networks.

Context Owners

[0073] According to one embodiment, a context owner may provide the hub. The term "context owner" will be used to refer to a service provider that provides a service network of other service providers. In one case, the context owner may use the hub and nodes to provide a virtual private network of itself and other service providers that provide an end-to-end value-added service or transaction. In this case, the hub may be located at the context owners web server, web site, or call center and the nodes may be located at the entry point into private enterprise networks of the other service providers. Advantageously, context providers may use the distributed control and management provided by the hub and nodes to provide control and management-added value to their service offerings.

[0074] Another type of context provider provides a service network of predetermined service providers associated with a multitude of transactional and service categories, any one of which may be selected and performed on the service network. For example, the context owner may be a dynamic yellow page provider resembling a search engine with the additional advantageous capability of being able to initiate a service transaction based on a search and involve a plurality of additional predetermined service providers in the transaction to add overall value to the transaction. In this way, a user of the dynamic transactional yellow pages may search for car dealers using the dynamic transactional yellow pages (e.g., search engine), locate a predetermined car dealer of the service network, be automatically connected with one of another predetermined banks of the service network, and be connected

with a selectable one of another predetermined number of insurance providers of the service network.

[0075] Alternatively, the context owner may be another context owner, such as a network service operators (e.g., AT&T, Sprint, MCI), an Internet service provider (e.g., AOL, UUNet, Netcom, PSINet), a portals (e.g., AOL, Yahoo!, CNET, enterprise portals), a virtual malls (e.g., Priceline, Shop@aol, ToysRUs.com), an e-marketplaces (e.g., Commerce One, Ariba), a direct merchant service (e.g., Bank of America, Fidelity, Vanguard, LL Bean, Amazon.com), an ASP (e.g., MGM/Blockbuster), an Internet brokerage firm (e.g., E*trade, Fidelity Investments), an extranet context owner (e.g., insurance industry, underwriters), an intranet context owner (e.g., a payroll processing center for a Fortune 1000 company connecting multiple departments and banks for timecard input, payroll deductions/withholding adjustments), a search engine (e.g., Yahoo!), and others.

[0076] Figure 7 shows conceptually illustrates a service network system showing a hub creating controlled links to multiple nodes, according to one embodiment. A client access device 710 accesses the Internet 720 and uses an IP address 730 to access a hub 740. By way of example, without limitation, the client requests or indicates to receive a service that involves interaction with node 1 760. The hub 740 is functionally interposed between the client access device 710 and node 1 760 and establishes link 750 to node 1 760. According to one embodiment, the link 750 is a controlled link that is controlled by the hub and supported or carried by the Internet 720 based on a predetermined IP address associated with node 1 760. After, simultaneously with, or before accessing the node 1 760, the hub establishes link 770 to node N 780 that is also associated with the service. The link 770 may be carried by the Internet 720 and based on a predetermined IP address, or may be carried on another facilities network typically compatible with client access device 710 if data entry by the client is needed, but this may not be necessary if only interaction with hub 740 or node 1 760 is needed by the node N 780 to perform its portion of the service. The link 770 may represent a hop that may be monitored and recorded by the hub 740 so that a hop-based fee may be charged from the node N 780.

[0077] Establishing the links 750, 770 are done under the control and management of the hub 740. This compares favorably with prior art approaches which provide hyperlinking and which would not be able to achieve centralized control and management of the service experience of the client access device. Advantageously, in this way, the client access devices service experience may be less like a visitor-center-type experience, such as through yellow pages or a search engine, in which the client is informed of a site and sent away to that site with loss of control over, and more like a supermarket-type experience in which control over the service experience of the client has not been lost, and the service control of the client may be managed, controlled, tracked, and otherwise improved.

[0078] Figure 8 shows a service network system 800 in which applications are closely connected to the service network. A node 810 includes a web server 815 a service network engine 820 and a gateway 825 directly connected to a hub 840 and the service network. The service network engine 820 represents node-side software to create and allow for the management of the service network. According to one embodiment, the service network engine 820 is node-side TransWeb™ Exchange software, available from WebXchange of Scotts Valley, California. The gateway allows access to applications 830 and data 835. This is in contrast to a prior art approach where the web server alone was directly connected to the Internet and applications were indirectly connected via the web server. The node 810 may access the hub 840 to connect with other service network connected entities 860, such as other nodes (within and between service networks), hubs, collaborating applications (which may be geographically dispersed), branch offices, and others. Thus, there may be hub-to-hub and node-to-node within and between service control centers, depending on the implementation.

[0079] Figure 9 illustrates in block diagram form a method 900, according to one embodiment, to perform a transaction via a service network. Typically, the method 900 will be implemented in logic that may include software, hardware or a combination of software and hardware.

[0080] The method 900 commences at block 910, and then proceeds to block 920, where a service network is accessed via a network entry point. According to one embodiment, a user connects to a web server (or a call center or cell site) running an exchange component, the user issues a request for a transactional application, the web server hands off the request to the exchange, the exchange activates a graphical user interface (GUI) to present user with a list of Point-of-Service (POSvc) transactional applications, and the user makes a selection from the POSvc application list. POSvc applications are transactional or service applications that are designed to incorporate and take advantage of the capabilities provided by the present invention.

[0081] The method 900 advances from block 920 to block 930, where switching to a transactional application is performed. Switching may include value-added network switching to local applications or components or remote applications or components and causing routing to those applications or components. Switching may also include flow control, prioritization of requests, and multiplexing. According to one embodiment, interconnected OSI model application layer software switches may perform the switching.

[0082] The method 900 advances from block 930 to block 940, where a route to a node is performed under the control of the hub. Routing may include performing multi-protocol routing to remote components or applications by using Simple Network Management Protocol (SNMP), TransWeb™ Management Protocol (TMP), or others. Traditional security features (e.g., RSA, SET1, SET2), and others are contemplated.

[0083] The method 900 advances from block 940 to block 950, where transaction processing is performed. This may include retrieving data from a data repository, such as by using TMP or another protocol.

[0084] A determination is made at decision block 960 whether another node is involved in the service. As stated above, the determination may include querying and receiving a response from the client and/or receiving an indication that another node is involved based on the prior transaction processing at block 950 and/or others. If yes is the determination 962 then processing loops through blocks 940-960 until no is the determination. Routing to the other nodes may be done with control and while

keeping the previous nodes involved in the transaction if they still have an interest in the transaction.

[0085] If no is the determination 964 then processing advances from decision block 960 to block 970 where transaction results are provided. The method 900 terminates at block 980.

[0086] Figure 10 conceptually illustrates components 1000 of a service network, according to one embodiment. A web browser access device 1010 accesses a web server 1020 that is functionally coupled with an exchange 1030. The exchange 1030 may reside on web server 104 or on any separate computer system that is at least connected with the Internet and capable of being accessed via an Internet address. Exchange 1030 creates and allows for the management (or distributed control) of a service network, operating within the boundaries of an IP-based facilities network. As shown, in one embodiment, the exchange 1030 contains an operator agent 1040, which may perform service network processing including interacting with a management manager such as those described elsewhere in the specification.

[0087] Together, the web server 1020, the exchange 1030, and the operator agent 1040 provide a web page 1050, one or more point-of-service (POSvc) applications 1060, VAN switch 1070, and object router 1080. According to one embodiment, the exchange 1030 displays an web page 1050 in the web browser 1010 including the list of POSvc applications 1060 that are accessible to the exchange 1030. A POSvc application is an application that can execute the type of service or transaction that the user may be interested in performing. By way of example, the list of one or more POSvc applications may be displayed in an HyperText Markup Language (HTML) GUI, a Virtual Reality Markup Language (VRML) GUI, a Java GUI, or another GUI.

[0088] Depending on the particular implementation, although they are shown as separate entities, the VAN switch 1070 and the router 1080 may be combined to form a router to provide multi-protocol object routing. In one embodiment, this multi-protocol object routing is provided via TransWeb™ Management Protocol (TMP), available from WebXchange Inc. of Scotts Valley California, which may incorporate

traditional security features (e.g., RSA, SET1, SET2, etc.). Alternatively, routing may be done using Simple Network Management Protocol (SNMP).

[0089] One embodiment of the present invention utilizes network accessible virtual information stores to perform routing. In one case, the virtual information stores are distributed on-line service information bases (DOLSIBS). Information entries and attributes in a DOLSIB virtual information store are associated with a networked object or component identity. The networked object identity identifies the information entries and attributes in the DOLSIB as individual networked objects, and each networked object is assigned a network reachable address (e.g., an Internet address). For example, the Internet address may be assigned based on the IP address of the node at which the networked object resides. Routing may be done using the DOLSIB and TMP or another protocol. In one case, TMP and a DOLSIB may be combined with Secure Sockets Layer (SSL), s-HTTP, Java, a component model (e.g., DCOM), the WinSock API, object request broker (ORB), or another object network layer to perform and manage object routing.

[0090] The VAN switch 1070 and object router 1080 will be described elsewhere in the specification. Thus, according to one embodiment, the exchange 1030 and an operator agent 1040, described in more detail elsewhere together perform the switching, object routing, application and service management functions according to one embodiment of the present invention.

[0091] Figure 11 conceptually illustrates a scheme 1100, according to one hierarchical tree-structure embodiment, to provide network addresses based on a unified numbering scheme for objects or components, which may be used in virtual information stores or DOLSIBs. A web server, which may be a node, has an exemplary network or Internet address 123.123.123.123. Object 1, which may be a Java applet, a COM object, or another object, has a network address based on the network address of the web server. In this particular example, the object 1's address is 123.123.123.123.1. Likewise, an object 2 and object 3 have network addresses 123.123.123.123.2 and 123.123.123.123.3, respectively. Similarly, network addresses may be provided for other objects, as desired. Thus, according to this exemplary

approach, objects may be addressed based on a hierarchical tree structure according to the node that they correspond to. Other network addressing schemes are contemplated.

[0092] The network or Internet address for each networked object essentially establishes the networked object as an accessible or "IP-reachable" node on the network or Internet. These network addresses may be used to represent the objects in a DOLSIB. For example, the network address 123.123.123.123.1 may be used to represent object 1 in the DOLSIB. The DOLSIB may also contain a along with a name, a syntax, and an encoding. The name is an administratively assigned object ID specifying an object type. The object type together with the object instance serves to uniquely identify a specific instantiation of the object. For example, if an object is information about models of cars, then one instance of that object would provide a user with information about a specific model of the car while another instance would provide information about a different model of the car. The syntax of an object type defines the abstract data structure corresponding to that object type. Encoding of objects defines how the object is represented by the object type syntax while being transmitted over the network. Then, TMP or another protocol may be used to uniquely identify and access these objects from the web server node, based on the network addresses recorded in the DOLSIB.

[0093] Figure 12 conceptually illustrates a service control center 1200, according to one embodiment, to provide verified services. A client access device 1205 accesses a hub 1210. The arrow 1206 conceptually represents the ordering and degree of verified completion of a service transaction. In particular, the arrow 1206 is unfilled representing that no stage of the service transaction has been verified completed as opposed to arrow 1246 which is filled and represents that all stages of the service transaction have been verified completed. For purposes of illustration, the arrow 1206 may conceptually represent a message or communication sent from the client access device 1205 to the hub 1210, although other back-and-forth and inter-party interactions between the shown client access device 1205, hub 1210, and the nodes 1215, 1225, and 1235 are contemplated for many other services.

[0094] To perform a service transaction that may be requested or indicated in a communication with the client access device 1205, the hub 1210 controllably connects with service provider node 1215. In this example, service provider node 1215 is a supplier selling products over the service network. The client access device 1205 indicates to purchase one model r100 at a cost of $100. The supplier 1215 connects and communicates with data source 1220 to obtain inventory data and update the inventory to reflect the purchase of one model R100 unit at a cost of $100. Arrow 1221 is partly filled to indicate that the requested model 100 is in inventory and was purchased. Status window 1222 indicates the purchase. A verified degree of completion of the service transaction is indicated by the difference in shading between arrows 1211 and 1223.

[0095] The hub 1210 determines that the purchasing portion of the service transaction has been verified completed and controllably connects with service provider node 1225, which in this example is a Visa node to bill payment to a Visa credit card account indicated by the client access device 1205. The Visa node 1225 communicates with data source 1230 based on, for example, a credit card number, to perform billing processing. Status window 1232 shows the billing. Completion of the billing portion of the service transaction is indicated by the difference in shading between arrows 1224 and 1233.

[0096] The hub 1210 determines that the billing portion of the service transaction has been verified completed and controllably connects with service provider node 1235, which in this example is a FedEx node 1235 to arrange delivery of the model R100. The FedEx node 1235 interacts with a data source 1240, based on delivery preference data supplied by the client access device 1205, to arrange delivery. Completion of the delivery portion of the service transaction is indicated by arrow 1243, which is entirely filled.

[0097] The hub 1210 determines that the purchasing, billing, and delivery portions of the service transaction are verified completed, as indicated in the status window 1245, and provides confirmation of the service transaction to the client access device 1205, as indicated in status window 1250. Advantageously, the transactional

control provided by the hub 1210 has allowed a multi-service provider value-added service to be provided to the client access device 1205, including verification of multiple transactional portions of the service. According to one embodiment, the hub 1210 is financially compensated by the nodes 1215, 1225, and 1235 based on a visit or hop to the node, a purchase, a purchase amount, and according to other desired criteria.

[0098] Figure 13 conceptually illustrates the Open System Interconnection (OSI) reference model that is useful to understanding embodiments of the present invention. The OSI model is a networking framework for implementing communication protocols in seven layers including a physical layer 1301, a data link layer 1302, a network layer 1303, a transport layer 1304, a session layer 1305, a presentation layer 1306, and an application layer 1307. Control is passed from the application layer 1307 located at one point in the network layer-by-layer to the physical layer 1301 over a network communication link to a second point in the network and back up the hierarchy from the physical layer 1301 to the application layer 1307. In one case each layer may communicate with its peer layer in another node through the use of a protocol.

[0099] Physical layer 1301 may transmit unstructured bits across a link. Data link layer may transmit chunks across the link and may perform check-summing to detect data corruption, orderly coordination of the use of shared media, and addressing when multiple systems are reachable. Network bridges may operate within data link layer 1302. Network layer 1303 may enables any pair of systems in the network to communicate with each other. Network layer 1303 may contain hardware units such as routers to handle routing, packet fragmentation, and reassembly of packets. Transport layer 1304 may establish a reliable communication stream between a pair of systems and deal with errors such as lost packets, duplicate packets, packet reordering and fragmentation. Session layer 1305 may offer services above the simple communication stream provided by transport layer 1304. These services may include dialog control and chaining. Presentation layer 1306 may provide a means by which OSI compliant applications can agree on representations for data.

[00100] The application layer 1307 typically defines the language and syntax that applications use to communicate. Application layer 1307 may provide a means for application programs to access the OSI environment. By way of example, an application on one computer system in a network uses application-layer prescribed commands to access or request data from an application located on another computer system of the network. Often the application layer 1307 is responsible for functions such as file management (e.g., opening, closing, reading and writing files), transferring files, transferring messages (e.g., email messages), executing jobs remotely, obtaining directory information about network computer systems, and other distributed computing applications. Application layer 1307 may include services such as file transfer, access and management services (FTAM), electronic mail and virtual terminal (VT) services.

[00101] According to one embodiment, the invention uses software conforming to the application layer 1307 of the OSI model to provide the service network by providing communication, control, and management of distributed software. For example, according to one embodiment, the routing switch is implemented to function within the application layer 1307 of the OSI model. Application layer routing may create an open channel for the management and the selective flow of data from remote databases on a network.

[00102] Figure 14 conceptually illustrates an exemplary layered architecture of a value-added network (VAN) switch 1400, according to one embodiment. VAN switch 1400 and other interconnected switches may be used to create an application network, backbone to provide the service network. The VAN switch 1400 includes a boundary service 1410, a switching service 1420, a management service 1430, and an application service 1440.

[00103] Boundary service 1410 may provide the interface between VAN switch 1400 and a facilities network and client access devices. Boundary service 1410 may also provide an interface to an on-line service provider. Using these interfaces, a client may use a client access device to connect to a local application, namely one

accessible via a local VAN switch, or be routed or "switched" to an application accessible via a remote VAN switch.

[00104] Switching service 1420 may perform a number of tasks including routing user connections to remote VAN switches, flow control, prioritization of requests, and multiplexing. Switching service 1420 may also facilitate open systems' connectivity with both the Internet (a public switched network) and private networks including back office networks, such as banking networks. Often, the switching service represents a core of the VAN switch 1400. According to one embodiment, the switching service 1420 is implemented as an OSI application layer switch.

[00105] Management service 1430 may contain tools that are used, such as by end users, to manage network resources including VAN switches like VAN switch 1400. For example, the tools may include Information Management Services (IMS) and application Network Management Services (NMS). Management service 1430 may also provide Operations, Administration, Maintenance & Provisioning (OAM&P) functions. For example, the functions may include security management, fault management, configuration management, performance management and billing management for the service network. Network management, such as provided by management service 1430, is another significant aspect of certain embodiments of the invention and may be used to add quality and value to the services provided.

[00106] Application service 1440 may contain application programs that provide customer services. For example, application service 1440 may include POSvc applications such as those discussed in Figure 10 and elsewhere. Other exemplary application programs that may be provided by application service 1440 include multi-media messaging, archival/retrieval management, directory services, data staging, conferencing, financial services, home banking, risk management and a variety of other vertical services. The applications service 1440 may contain applications having design features that allow them to conform to standards related to performance, reliability, maintenance and ability to handle expected traffic volume. Depending on the type of service, the characteristics of the network elements may differ. Typically, application service 1440 will provide a number of functions including

communications services for both management and end users of the network and control for the user over the user's environment.

[00107] Figure 15 conceptually illustrates a VAN switch 1500, according to one embodiment. The exemplary VAN switch 1500 contains an exchange 1520, and a management agent 1560 that are potentially geographically distributed over the Internet 1510. The exchange 1520 and the management agent 1560 may take on different roles as desired, including peer-to-peer, client-server or master-slave roles. Management manager 1550 may reside on a separate computer system either on the Internet 1510 or anywhere where the Internet 1510 connects with another computer system or network. Management manager 1550 may interact with an operator agent associated with the exchange 1520. In alternate embodiments, two or more of the components shown may reside on the same computer system or location in the Internet 1510.

[00108] An object router may be used to controllably route to networked entities such as computer systems, applications, objects, and data. The object router may allow for the transparent completion of service transactions involving distributed applications and software components without the programmer needing to know whether networked entities are local or remote. The router may be able to automatically determine this, such as based on looking up a network address of a relevant entity and using correct operations compatible with the type of entity. An object router may include a library to provide support for the application programming interfaces (APIs) to remotely access an object, its data, and its functions in an object network. This interface may provide a skeleton class to contain the functionality of the object and corresponding or counterpart stub class to allow remote access of the object. A stub and a skeleton may be functionally coupled together. For example, a stub may be installed on a client computer system and a corresponding skeleton installed on a server computer system and in combination they interoperate to allow a remote procedure or method call. In one case the stub may declare itself and its parameters. Arguments to the function may be specified in a meta file and a type of the argument may be specified by value or by reference. The object router may allow

for new data types to be constructed, using the basic data types of the programming language used in the particular embodiment: int and String. Single simple inheritance classes may be constructed and then used as data members or pointer within another meta class. Typically, the router will be implemented in a programming environment and language that is object-oriented and allows for distributed computing, such as C++, Java, and a component model. However other embodiments are also contemplated.

[00109] Before continuing with the detailed explanation of the present invention and various exemplary embodiments of the present invention, it may be helpful to briefly explain some terms, without limitation, that will be used in the discussion below. These explanations are provided to facilitate understanding of the following text, rather than to limit the invention. The term "abstract class" will be used to refer to a C++ class that does not have all virtual functions defined. The term "class" will be used to refer to typically a C++/Java data structure definition that defines both the data and the functions. The term "interface" is a Java term similar to the C++ abstract class. A "meta-compiler" translates a higher-level "meta-language" (e.g., WebX, available from WebXchange) from the "meta-file" into a lower-level language (e.g., C++) output file for and before giving to a traditional compiler. The software may be compiled under a version of Windows NT using a Microsoft Visual C++ version compiler based on the wx.lib and the Rogue Wave libraries, available from Rogue Wave of XXX, XXX, Tools++, Net++ and Threads++. Other software platforms are contemplated. The term "object" may be used to refer to a C++/Java data structure instance that is defined by a class.

[00110] Figure 16 conceptually illustrates software layers of an object router 1600, according to one embodiment. The layers include a transport layer 1610, a line protocol layer 1620, a marshalling / serialization layer 1630, a connection management layer 1640, an exception handling / thread rendezvous layer 1650, a class abstraction/stub & skeleton layer 1660, and an distributed object model layer 1670.

[00111] A meta compiler 1680 may be provided for use with the layers 1660 and 1680 The meta compiler 1680 will be used broadly to refer to an automated

mechanism to code features based on structured typically concise definitions. For example, the meta compiler 1680 may take a definition file and substantially automatically create the object identity, data serialization, data marshaling, string execution, abstract base class, and the stub/skeleton multiple inheritance. Advantageously, such automated coding may improve the efficiency of the implementation and may reduce errors. Of course, coding may be performed manually without such a meta compiler, although such implementations are expected to be more laborious, expensive, and prone to error.

[00112] The meta compiler 1680 may use a Tool Command Language (TCL) program or a similar program or encoding. TCL is an interpreted script language that may be used to develop applications such as GUIs, prototypes, CGI scripts, and others. TCL may provide an interface into C, C++, and other compiled applications. The application is compiled with TCL functions, which provide a bi-directional path between TCL scripts and the executable programs. TCL provides a way to "glue" program modules together. TCL may also come as TCL/TookKit (TCL/Tk), which provides a GUI toolkit to create GUIs. Scheme, Perl, and Python have incorporated elements of TCL/Tk. According to one embodiment, the meta compiler 1680 is the rme2c meta compiler discussed elsewhere in the present application.

[00113] In one embodiment, the meta compiler is run by the command rme2c<classname>, where the classname is the base class (e.g., Account). The Account.rme file as well as other parent definitions should desirably be in the same directory. The object router TCL files are found under Wx/Util. These files parse the description file and produce the six C++ output files. Often, the syntax of the meta compiler should be adhered to closely. Blank lines and lines beginning with a pound sign "#" are considered comments. The following words may be reserved:

- include [c*,h*,j*,*base,*skel,*stub] To add code or verbose code to one of six files: cbase, hbase, cstub, hstub, cskel or hskel. The programmer may also specify all "c" files, "h" files, or "base", "skel" or "stub" files.
- endinclude - to end the verbose inclusion.

- header <class> - to indicate that the compiler will wait for the header section to complete.
- beginclass <class> [<parent>] - to indicate the class name and any parent.
- begindata - to signals the begin of the data section
- data [<perm>] <type> <name> - to provide a data definition statement.
- enddata - to end the data section.
- beginmethod - to begin the method section.
- method [const] [<penn>] <return_type> <name> [{<arg1_type> <arg1_name>}, . . .] - to define a method.
- endmethod - to end the method section.
- endclass - to end the class definition; typically this is the last statement in the file.

[00114] With reference to Figures 16, and portions of Figures 18 and 21-22, an exemplary object router will be described that provides distributed transactional services based on controlled connection and communication between distributed software objects. During this discussion, details are given, including the model represented by Figures 18 and 21-22 and particular objects, methods, syntax, convention, and other particulars that are useful to illustrate operation of certain embodiments but which are not needed. Those having an ordinary level of skill in the art will appreciate that there are alternative implementations that take entirely different modeling approaches compared with the models shown in Figures 18 and 21-22. They will also appreciate that the syntax is dependent upon the elected programming convention and may change for non-C++, non-Java, and non-object oriented environments. Accordingly, while the particulars are useful for illustration, they should be viewed in that illustrative sense rather than in a limiting sense.

[00115] A detailed discussion of an exemplary object router is provided without limitation to further illustrate operation of an object router according to certain embodiments. In the following discussion, the reader is respectfully directed to Figures 18 and 21-22 and associated text for further illustration and discussion of the

characteristics and structure of the classes and objects used by the exemplary object router.

[00116] The object router typically includes functionality to determine an object identity for a networked object in order to communicate with the object. The identity may be determined from a library that stores identities for many such networked objects. In one case, the object identity may be determined by using a WxObject in a wx.lib library. In such a case each new class "XYZ" may add a XYZ_ClassID, XYZStub_ClassID and XYZSkel_ClassID to the Wx/ClassID.h file. Based on the object identity, the object router may transparently determine whether the object is local (e.g., a skeleton on a server) or remote (e.g., a stub on a client or an object on another server). The object router may also determine the argument parameters and object serialization.

[00117] The object router typically uses some protocol to communicate with remote computer systems and software. Communicating may include transferring objects, parameters, and data. Often a network line protocol/TCP will be used. The WxRemotePDU shown in Figure 22 includes a plurality of parameters and methods to provide a protocol data unit (PDU) that conforms to the protocol. The objects, parameters and data may be serialized onto a network stream that is sent to the remote computer system. The persistence nature of RWCOLLECTABLE shown in Figure 21 may provide the data serialization to transmit the data. Often, at least a portion of the code to perform these functions will be generated by a meta compiler based on base classes.

[00118] The object router may also perform data marshalling. Data marshalling may include checking parameters that are passed as arguments to methods to determine if any parameters are missing or wrong, and may include throwing an exception or otherwise signaling if a parameter is missing or wrong. This may be provided by a meta compiler in the stub and skeleton.

[00119] The object router may block a thread during network transmission and reception by using a different thread to perform the actual network activity and control the calling thread status. Advantageously, this may allow a remote object to be called

similarly to a local object. In one case, a Rogue Wave RWCondition class in the WxRemotePDU class may perform this function.

[00120] The object router may use string execution to allow an ASCII string representation to call a method. This may provide a useful, simple, and unique means of calling a class method that may also be used directly by a programmer. A meta compiler in the base class may create this data marshalling.

[00121] The object router may also perform reference counting on local and/or remote objects. This may avoid time consuming malloc's, free's, and confusing details of which function is responsible for deleting which object. The object router may use such reference counting to deal with WxRemoteObject types. The programmer may also use reference counting. Typically, a WxRemoteObject child should not be destroyed using delete, and an exception may be thrown if this is tried, but rather the WxRemoteObject child should be destroyed by using the member function ol_unreference( ). Also, if a user stores a copy of a WxRemoteObject child, the method ol_reference( ) should be called to prevent the object from being destroyed by some other user or method. WxRemoteObjectInt may provide this interface.

[00122] The object router may use an abstract base class and multiple inheritance according to certain embodiments. Advantageously, the abstract base class may allow interaction with a local or remote object without knowing its location. This base class may be the parent of both the stub and the skeleton, which may be inherited from the abstract base class and from their respective object layer classes. This allows them to inherit the functionality of the base class as well as the object layer routines. Often, the inheritance will be provided by the meta compiler.

[00123] The object router may also include at least an interface to a user-defined data model to provide a basis of the object layer to the next level up the software layers. The user-defined data model may include a set of user-created classes built around and on top of the object router APIs. This foundation appears local to the programmers using the object layer even though it may be remote.

[00124] Figure 17 conceptually illustrates data model integration 1700 for an object router of one embodiment. The object model 1700 represents one breakdown

of base classes into inherited or derived classes, although other embodiments are contemplated. For convenience, the components will be referred to by name (e.g., WxObject) rather than by number.

[00125] The object model 1700 comprises base classes WxObject and WxRemoteObjectInt from which other classes and objects derive. The derived components are either object layer objects 1720 (e.g., Object, WxRemoteObject, WxReference, WxRemoteSkel, WxRemoteStub) or data model objects 1740 (e.g., WxName, WxNameStub, and WxNameSkel).

[00126] A programmer that is creating transactional objects is likely to work closely with the base class WxRemoteObject and/or WxRemoteObjectInt. WxRemoteObjectInt is an abstract base class that may contain all of the member methods and data access components as well as support functions to provide a uniform interface to behave as a WxRemoteObject. For example, this may include (a) WxStringExecution to execute any method using a simple ASCII string and object type data, (b) WxLock to provide a thread synchronization mechanism, (c) WxFlags to provide a simple and consistent Boolean flag variable, (d) reference counts to allow sharing and manage ownership concerns, (e) conversions between OBJECT, WxRemoteReference, WxRemoteStub and WxRemoteSkel types, and others as desired. As shown, both the object layer objects 1720 and the data model objects 1740 inherit an interface specification WxRemoteObjectInt.

[00127] WxName is a new data model object 1740 that contains one data member "name" which is a string. The meta compiler may automatically create two access components for this data member, namely Get_Name and Set_Name. The meta compiler may also create the server and client versions of this data object.

[00128] Typically the skeleton is used to represent the server side of the object router. For example, WxNameSkel is a class that is derived off the abstract base class WxName and WxRemoteSkel. Often the programmer defines or customizes the methods, except for the data access components, for the skeleton, since this is the actual object embodying the business or transactional methods. Real instances of the class may be created with the suffix "skel".

[00129] The stub represents the client or remote side of an object for the object router. As with the skeleton, the stub too is derived off the abstract base class WxName and an object layer class WxRemoteStub. Typically, the meta compiler will generate all methods for the stub.

[00130] Without limitation, the use of certain conventions and codes (e.g., prefixes, suffixes, etc.) may be used to improve certain implementations. A partial list of exemplary conventions and codes is provided below. Those having an ordinary level of skill in the art will appreciate that the exemplary conventions and codes are not needed to implement the invention. They will also appreciate that numerous other conventions and codes may be conceived to improve certain other aspects.

- The StringExecutionInt class may prepend all of its member functions with "se_".
- The object layer classes may prepend their member functions with "ol_" to avoid name-bashing with derived classes that are built using the base classes created by the meta compiler.
- Member data in the skeleton may be prefixed with "_" as a reminder and indicator that the data is local to the object and usually protected.
- To deal with synchronization issues, any modification of local member data within a skeleton should be guarded by ReadLockGuard or WriteLockGuard when accessed since other threads may be sharing this data.
- Skeletons may be suffixed with "skel" and stubs may be appended with "stub".
- WxRemoteObject derived data may be passed with a pointer.
- To indicate who has ownership of the data, the suffixes "_ref" and "_val" may be added by the meta compiler to indicate if the data is passed by value or by reference. If it is passed by reference, the function may then return a pointer that has had the reference count incremented. If it is passed by value, the data may be copied from the original source and the receiver may unreference this using DeleteObject.
- To indicate if a data member is passed by reference, an asterisk (i.e., *) may be appended to the data type in the "data" declaration section of the object router

meta file. Similarly, this may be done for return types in the "method" section and for arguments to methods.

- When data is passed by value into a function, it will be proceded with "const" to signify that the object is not to be changed.
- The header file Wx/ClassID.h may contain all ClassIDs for the base classes and their stubs and skeletons. Object Ids may be placed in this (e.g., by the programmer) before running the object router meta compiler.
- The CC and HH files may be included from the object router meta file to add additional functionality to the base, stub or skeleton. For example, if a function void xyz( ) is added to the XYZ skeleton class it may then be added to the XYZ.rme file:

```
include hskel
void xyz( );
endinclude
include cskel
#include "XYZSkel.cc"
endinclude
```

  This will then include the simple declaration "void xyz( )" into the header for the skeleton and also include the definition for xyz( ) from the XYZSkel.cc file.
- The suffixes "cc" and "hh" may be used rather than "cpp" and "h" since the object router meta compiler uses those suffixes for the final XYZ files.
- The CPP and H files are automatically generated by the object router meta compiler for the base, skeleton and stub. Desirably, the programmer should not edit the cpp or h files directly. Rather, the programmer should modify the rme file and recompile.
- Strings used in the object layer may be passed either by value using RwcString and "const char*" or by reference using the RwCollectableString (also known as "string"). In some cases, the programmer knows which version is most desirable: pointer or static object. Based on programmer need, the programmer

can choose either the function foo( ) which returns the string by value or foo_ptr( ), which calls the same function but returns a copy of the string on the heap as a pointer.

- Two data access components may be automatically created for each data member, namely "get" and "set". There may be different operation for different types of data such as integer, string, and others. The integer case is the simplest and creates member functions int getxyz( ) const and void set_xyz(int). The string case has been mentioned elsewhere, and creates three methods: RwcString get_xyz( ) const, String *get_xyz_ptr( ) and set_xyz(const char*). The case WxRemoteObject by value creates two functions: XYZ* get_xyz_val( ) const and void set_xyz(const XYZ*). The case WxRemoteObject by reference also creates two functions, XYZ* get_xyz_ref( ) const and void set_xyz(XYZ*). This also assumes that the "set" function will retain a copy of the object.

[00131] Figure 18 conceptually illustrates a banking service transaction 1800 involving a single bank service provider, according to one embodiment. The banking transaction 1800 includes a client HTML browser 1805 accessing a web server file system 1810 associated with the bank. The web server file system 1810 returns a bank introductory web page 1815 that is displayed via the browser 1805. The web page 1815 may include any desired content as well as a transactional request mechanism 1816 (in this case [Access Account]). The user selects the request mechanism 1816 indicating a desire to perform the banking transaction.

[00132] In response to the selection, the web server 1810 starts an applet 1820 that runs in the web browser 1805. The applet 1820 registers with the object router. The object router may determine the identification and network location of one or more objects associated with the transaction. The object router may assist with creating stub 1830 on a computer system 1825, which may be the computer system running the browser 1805 or another computer system. According to one embodiment, the computer system 1825 may be a hub.

[00133] Via the stub 1830 a connection is made to a server 1835 containing a skeleton object 1840 associated with the bank transaction. Thus, once the stub is received, the user can then look up bank accounts as if local to the skeleton on the server side. The skeleton object 1840 presents transactional data 1850 to the user. Often, the transactional data 1850 will include a field for data entry, such as the ID and Pin fields shown. The user may enter data into these data fields and return the entries to the skeleton 1840 via the stub 1830. Based on these entries the skeleton 1840 may perform transaction processing. Transaction processing may include connecting with other local objects such as a user-specific Bob's account object 1845 and non-user specific Joe's account object 1846. In this case, user-specific data may be obtained from Bob's account object 1845 and returned to the browser 1805 via the skeleton 1840 and stub 1830 as transaction data 1860 including deposit interaction field 1862 and withdraw field 1864.

[00134] A stub object 1865 associated with the transaction data 1860 may be established at the computer system 1825 to perform transactions associated with the interactions 1862 and 1864. The user may then enter an amount into deposit interaction 1862, which then activates a deposit of said amount into Bob's account object 1845 via object 1865. Of course, in other implementations this could be handled differently. For example, such operations could be performed by a stub 1830, which also includes the functionality described for stub 1865.

[00135] Advantageously, the user was able to receive controlled banking transaction processing. Other embodiments are contemplated, including more complicated and interactive single service provider transactions (e.g., in which more arrows are bi-directional) and involving multiple service providers. The later case of multiple service providers will be shown and described for Figure 20. However, first it may be useful to provide further implementation details that may be used to implement the banking transaction 1800. Other details and corresponding details for the other embodiments discussed herein will be apparent to those having an ordinary level of skill in the art based on the present disclosure.

[00136] A programmer may begin by creating a definition file describing the WxBank and WxBankAcount objects. Typically this will be written in a simple language, such as TCL, which may be parsed by the rme2v meta compiler. For example, the WxBankAccount file may be written as:

```
include cskel
# INCLUDE ANY ADDITIONAL C CODE FOR THE SKELETON
#include "WxBankAccountSkel.cc"
endinclude

beginclass WxBankAccount

begindata
# PUT MEMBER DATA HERE
data int balance
end data

beginmethod
# PUT MEMBER METHODS HERE
method void deposit {int x}
method void withdraw {int x}
endmethod

endclass
```

[00137] This class may contain methods and data. In this case, the data may be an integer describing the amount of money the account holds. Deposit and withdraw methods may increment or decrement the integer amount as follows:

```
void WxBankAccountSkel::deposit(int x) {
 WriteLockGuard lock(wxlock());
```

```
_balance += x;
}
void WxBankAccountSkel::withdraw(int x) {
 WriteLockGuard lock(wxlock());
 _balance -= x;
}
```

[00138] Notice that the programmer should provide thread locking. For example, by adding the statement, WriteLockGuard lock(wxlock( )) to each desired method. Note that when the method is locked, no other locked methods that include any object-layer defined data access components may be called. The above file (WxBankAccountSkel.cc) defines the skeleton methods. The stub methods are typically defined by the rme2c meta compiler.

[00139] The Bank.rme file may be represented by the following code:

```
# INCLUDE ANY ADDITIONAL C CODE FOR THE SKELETON
include cskel
#include "WxBankSkel.cc"
endinclude

# INCLUDE ALL H FILES FOR DATA TYPES USED IN ALL C FILES
include c
#include "WxBankAccountSkel.h"
#include "WxBankAccountStub.h"
endinclude
beginclass WxBank

# PUT MEMBER DATA HERE
begindata
enddata
```

```
# PUT MEMBER METHODS HERE
beginmethod
method const WxBankAccount* getAccount {int id} {int pin}
endmethod
endclass
```

[00140] This file, when processed by rme2c will create six files: WxBank.h, WxBank.cpp, WxBankStub.h WxBankStub.cpp, WxBankSkel.h and WxBankSkel.cpp. These six files describe the operation and remote execution of the WxBank object. Since there is no data, no data access components will be generated. The method "getaccount" is defined as follows: method const WxBankAccount* getaccount {int id} {mint pin}. The keyword "const" identifies that this method will not change the object data. The next keyword is the returned object "WxBankAccount*". The asterisk indicates that the object will be passed by reference. The "getaccount" is the actual method name. Two parameters of the method are provided next in braces. Each parameter is provided in braces with a data type followed by a parameter name. In this case there are two integer parameters with name id and pin.

[00141] The programmer may describe any additional functionality for the operation of this object and the definitions of the skeleton methods in the WxBankSkel.cc file. The WxBankSkel.cc file may contain:

```
WxBankAccount* bob = 0;
WxBankAccount* WxBankSkel::getAccount_ref(int id, int pin) const {
 if(bob) bob->ol_reference();
 return bob;
 }
```

[00142] This is a simple example in which getAccount returns Bob's account. Note that the actual method name is "getAccount_ref" with "ref" appended since this method will return an object by reference. Also, notice that before simply returning

the global variable Bob, the reference count is incremented since getaccount is passing a new reference.

[00143] Typically the skeletons are created in the server side. Then the skeletons may be registered in the name server, as indicated by the following exemplary code:

```
extern WxBankAccount* bob; // global used by Bank::getAccount()
void main(int argc, char* argv) {
    RWWinSockInfo winsock; // initialize the socket library
    WxRemoteConnectionServer s; // create the socket server
    WxBank bofa; // create a bank
    WxBankAccount joe; // create joe's account
    joe.set_balance(0); // with a $0 balance
    bob = new WxBankAccount(); // create bob's account
    bob->set_balance(10000); // with a $100 balance
    bob->deposit(20000); // then, deposit $200.
    // register bofa with a global name --- after everything else is done!
    bofa.set_ol_name(new String("BofA"));
    // start the connection server receiver
    RWThread server =
     rwMakeThreadFunction(
      s,&WxRemoteConnectionServer::run,(RWBarrier*)0);
    server.start();
    server.join();
}
```

[00144] The client may have the following exemplary code:

```
// create a global function which is called from a RogueWave thread.
void async() {
    WxRemoteConnectionMonitor monitor;
```

```
    WxRemoteClient* local = monitor.client("localhost");
    WxBank* bofa = LOOKUP(WxBank,"BofA",local);
    WxBankAccount* bob = bofa->getAccount(10,20); // arguments are
    dummy
    cout << "bob's account balance is (should be 30000):"
     << bob->get_balance() << endl;
    bob->withdraw(5000); // withdraw $50.
    cout << "bob's new balance is "<<bob->get_balance() << endl;
}
void main(int argc, char* argv) {
    WxRemoteObject::initializeStringExecutionTables();
    WxBankSkel::wxClassHierarchy();
    WxBankStub::wxClassHierarchy();
    WxBankAccountSkel::wxClassHierarchy();
    WxBankAccountStub::wxClassHierarchy();
    RWWinSockInfo winsock;
    // start the RogueWave thread - and wait until it exits
    RWThread thread = rwMakeThreadFunction(async);
    thread.start();
    thread.join();
}
```

[00145] Advantageously, in this way the programmer does not have to know, nor care, whether the object "bob" is local or remote.

[00146] Figure 19 conceptually illustrates a multi-service provider transaction 1900, according to one embodiment. In this particular example, a web browser client 1902 accesses a remote hub 1904 that serves as a network entry point.

[00147] The hub 1904 includes a greeter 1915, which may be software or a dedicated server. In this case, the greeter 1906 contains a web page 1908 containing HTML code and an applet 1910. The web page 1908 presents a window 1912 in the

browser client 1902 including text 1914 and a selection mechanism 1916 to indicate a particular transaction (e.g., in this case to purchase product 1).

[00148] In response to a selection of the mechanism 1916 the applet 1910 starts running in the client 1902 to present a transaction window 1918 and a VAN switch (not shown) may switch to a particular transactional application associated with the mechanism 1916. This may include registering with an object router 1920. The router 1920 may then route to a first node 1922 including a supplier object 1924 and a product object 1926 which may then return window 1928 including a cost $100 for product 1 and payment options including a mechanism 1930 to allow payment from a particular bank's bank account.

[00149] In response to selection of the mechanism 1930 the router 1920 routes to a second node 1940 including a bank object 1942 associated with the bank. The bank object 1942 returns a window 1946 including an ID entry mechanism 1948 and a Personal Identification Number (PIN) entry mechanism 1950.

[00150] In response to submission of a corresponding ID and PIN the object router 1920 routes to the bank object 1942 and an account object 1944 corresponding to the ID and PIN. The account object 1944 returns a window 1960 including account corresponding to the client of the ID and PIN including a balance of $4000 and an electronic payment option mechanism 1962.

[00151] An account stub 1964 may also be activated or transferred to the hub 1904 to correspond and interface to functions associated with the account object 1944. In response to selection of the payment option mechanism 1962 the account stub 1964, the account object 1944 and the supplier object 1924 may interact and process so that the purchase price of $100 for the product 1 is paid from an account of the client to an account of the supplier.

[00152] Numerous variations and alternative embodiments are also contemplated for a multi-service provider transaction. For example, several single-directional arrows have been shown for purposes of clarity, however any or all of these arrows could represent bi-directional communication. Additionally, certain objects (e.g., supplier object 1924 and product object 1926) could be combined, or

further subdivided into additional objects. Accordingly, the example is to be viewed in an illustrative rather than a restrictive sense.

[00153] Figure 20 conceptually illustrates a banking transaction 2000, according to another embodiment. As shown, stub components 2005, 2010 may be located on a first computer system 2015 and corresponding skeleton components 2050, 2055, 2060 may be located on a second computer system 2065. These components may behave as described elsewhere in the present application.

[00154] The first computer system 2015 may have a WXREMOTECLIENT object 2020 to request access into the service network. The second computer system 2065 may have a WxRemoteConnectionServer object to receive the request and serve as the entry point into the service network. This is an object of the main class of the server side of the object router. The WxRemoteConnectionServer then connects the WxRemoteClient with the WxRemoteServer. Then, the WxRemoteClient and WxRemoteServer may communicate by a TCP socket 2040. By way of analogy, this is similar to a person (WxRemoteClient) dialing up a telephone operator (WxRemoteConnectionServer) and the operator directing the call to the correct person (WxRemoteServer).

[00155] Figures 21 and 22 conceptually illustrate a class diagram showing the classes and functions of an object router of one embodiment. The schematic 2100 shows the relationship between Figures 21 and 22. A brief discussion of the classes and functions will be provided to further illustrate operation of an object router. Those having an ordinary level of skill in the art will appreciate that other object routers, classes, class models, and functions are contemplated.

WXREMOTEOBJECTINT

[00156] This is an interface typically included in all remote objects. The interface usually contains several abstract class definitions, including WxStringExecutionInt, WxLock and WxFlags. It may also define methods that are used by all or multiple remote objects.

RWBoolean ol_isValid( ) const

This may be tested if the programmer does not know if this object is local or if the connection is established. This will return TRUE if the object is local or it has a connection to the remote object.

unsigned get_ol_referenceCnt( ) const

This returns the number of pointers outstanding for this object. Typically, if garbage collection is enabled, this object will automatically be destroyed when the referenceCnt reaches zero.

WxReferenceId get_ol_referenceiD( ) const

This is the remote referenceId for this object. This WxReferenceId uniquely tags an object instance on the server for the established connection. This is not a well-known name in the sense that it is not guaranteed to be the same with a difference connection.

unsigned ol_reference( ) const

This increments the number of references outstanding. Typically this will be performed whenever a new copy of the pointer is stored.

void ol_unreference( ) const

This decrements the reference count and should be called instead of delete.

Object* ol_object( ) const

This type casts this instance to an RWCollectable pointer.

WxRemoteStub* ol_stub( ) const

This will return a stub for this object. If the object is local it will create a stub, otherwise if this is already a stub it will increment the reference count.

WxRemoteSkel* ol_skel( ) const

This will return a skeleton for this object. If this is a skeleton it simply increments the reference count. If this is a stub it will create a new skeleton, copy the data, and return it.

WxRemoteReference* ol_remoteReference( ) const
This will create a WxRemoteReference object that is used for serialization.

WxRemoteObject
This is the actual first level implementation of the above interface and adds String Execution to the above functions. Typically all of the router objects are derived from this object.

WxRemoteReference
This is a type of network "pointer" which indicates where the actual skeleton object resides. It contains the following data: RWInetHost host; int port; RWClassID classID; and WxReferenceId referenceId. The port and host uniquely specify the socket for the WxRemoteConnectionServer. The referenceId uniquely specifies which object on the WxRemoteConnectionServer is pointed to. The classID is used to construct a local stub object.

WxRemoteStub
All stubs are derived from this object and the abstract base object for the class. This object provides some interfaces to the object router library that is used by the meta compiler.

WxRemotePDU*_ol_execute(WxRemotePDU* pdu) const

This will block until ol execution is finished. It will take the pre-formatted PDU.

WxMarshalId_ol_send(WxRemotePDU* pdu) const

This is a non-blocking remote execution, which returns a WxMarshalId that may be used to receive the result.

WxRemotePDU*_ol_peek(WxMarshalId id) const

This checks if the PDU id is returned from execution.

WxRemotePDU*_ol_receive(WxMarshalId id) const

This blocks until the PDU is returned.

WxRemoteClient* _ol_connect( ) const

This ensures the connection to the other side is established.

WxRemoteSkel

All skeletons may be derived off this object and the abstract base for the class. This object provides the interface ol_methodPDU( ) for the meta compiler to the object router.

WxRemotePDU

This is the actual data packet sent across the network. The data in this are:

WxMarshalId id

This is the PDU packet number, typically a monotonically increasing integer to uniquely identify the packet.

unsigned flags

These are option flags to modify the execution of this protocol. The flags may include:

Syn - this will perform synchronous execution of the packet at the server (no threads).

NoMarshal - this is an unconfirmed execution similar to UDP.

Log - this will log this request

Response - this indicates that the PDU is a response

Val - this indicates that the result should be a value rather than a reference.

unsigned type
This is one of several known protocol operations:

Disconnect - close the connection between WxRemoteClient and WxRemoteServer

Error - an error occurred in processing the request

Result - a packet containing the result of a request

Lookup - a request to find a WxRemoteReference based on a well-known name in the WxRemoteNameServer

Ping - a request for the server to send a Pong back.

Pong - a response from the server to the client to a Ping

Method - a request to execute the command on the server

Unreference - a request to decrement a reference count.

Reference - a request to increment a reference count.

RWCString cmd
This is an ASCII string command to execute on the remote server. This is the "name" in a Name-Value pair.

WxReferenceId referenceId
This is the object WxReferenceId on the server to uniquely identify the object of this PDU.

Vector* data
This is the data for a method execution. This is the "value" in a Name-Value pair.

WxRemoteConnectionServer
This is the main class on the server side of the object router that may serve as an entry point into the system for a WxRemoteClient requesting access. It may connect the client with the correct objects to perform the transaction.

WxRemoteConnectionMonitor
This is the main class on the client side of the object router that may serve as an entry point into the system for a connection and to create a WxRemoteClient for a particular destination. By way of analogy, this is similar to a phone operator who directs outbound calls to the correct person. That person, in this analogy, is the WxRemoteConnectionServer.

WxRemoteServer

This is a component on the server side of the client-server communication channel to process each inbound request. The WxRemoteServer will spawn a new thread for each WxRemotePDU method packet. There is one WxRemoteServer for a WxRemoteClient.

WxRemoteClient

This is a component on the client side of the client-server communication channel to send each outbound request and rendezvous with the inbound response. There is one WxRemoteClient for a WxRemoteServer.

WxRemoteError

This is the class that is thrown by both the client and server side when an error is detected. These may be fatal and non-recoverable.

WxRemoteException

This is a class that is thrown by both the client and server side when an exception is detected. These may not be fatal and the programmer may provide recovery code, as desired.

[00157] Figure 23 conceptually illustrates a timing diagram for an object router, according to one embodiment. The timing diagram represents the startup of the object-layer on both the client and server sides and shows the operation and timing of objects and threads. It also demonstrates a lookup("root") call from the client and a later getname( ) method call on the root object. The vertical lines represent the objects, data, and functions used in the client and server sides. The horizontal lines represent the threads in the operating system. Different threads have different line patterns. There are two threads shown on the client side (left) and there are three on the server side (right). Also, the dotted "write" line represents TCP traffic between the two machines (client and server).

Distributed Online Service Information Bases

[00158] Certain embodiments of the present invention may use a virtual information store suitable for a network. Without limitation, a specific type of virtual information store, referred to as a Dynamic Distributed Online Service Information Base (dynamic DOLSIB), will be discussed in greater detail. Other virtual information stores are contemplated.

[00159] The object router may use the dynamic DOLSIB to perform routing. For example, the object router may access the dynamic DOLSIB to obtain information about distributed software objects that has been recorded in the DOLSIB. Typically, the enterprise will be customized for each merchant. The following sections cover an overview of the architecture for the DOLSIB and a uniform interface that allows the service provider to provide a customized interface for the business objects using a simple Extended Finite State Machine (EFSM) or DOLSIB language. A library is also described that provides the core parser and interpreter for the DOLSIB. This library may also serve as the base class for business and management objects that will interface with the enterprise interface.

[00160] Before continuing with the detailed explanation of the present invention and various exemplary embodiments of the present invention, it may be helpful to briefly explain some terms, without limitation, that will be used in the discussion below. These explanations are provided to facilitate understanding of the following text, rather than to limit the invention. The term "state" will be used to refer to the set of values describing the current position of the machine if it has memory. The term "transition" will be used to refer to the action and state change performed by the machine after receiving an event. The term "event" will be used to refer to the inbound trigger that causes the machine to perform some transition. The term "action" will be used to refer to the output of the machine as a result of a transition. The term "diagram" will be used to refer to a complete finite state machine description containing states and transitions.

[00161] The architecture of a business object may at least conceptually comprise four parts, including: (1) the Extended Finite State Machine (EFSM)

DOLSIB in the CoreBusinessObject or Management Object (C++), (2) the object router interface for the business or management object to the DOLSIB. (C++), (3) the enterprise interface protocol (specification), and (4) the DOLSIB instructions for the business or management object (EISM). The first part (DOLSIB and CoreBusinessObject or management object) may be built only once and may be part of the object router library. The second part may be built as a common business object and should be generic enough to be configurable for different merchants. The third part is a specification that may be written by the merchant for his own enterprise interface. The fourth part may be configurable during runtime for different business or management objects.

[00162] The following sections further discuss the DOLSIB, the language and grammar of the DOLSIB, and the CoreBusinessObject or management object. Specific examples, in this case banking examples, illustrate different service provider enterprise interfaces.

[00163] Figure 24 conceptually illustrates a simple Finite State Machine (FSM) 2400 that is useful for understanding concepts of a DOLSIB and an Extended FSM (EFSM). The FSM 2400 includes the two states S = {A, B} and the two transitions T = {t1, t2}. A transition t consists of an initial state ts, an event e that triggers an action a, and a final state tf. The transitions can be described as t1 = (A, X, Y, B) and t2 = (B, U, V, A). The transition t1 from state A to state B is triggered by an event X and causes an action Y. Likewise, the transition t2 from state B to state A is triggered by an event U and causes an action V. If the FSM 2400 is in state A and receives any event besides X, it will remain in state A. In this way, the FSM 2400 responds to valid events having predetermined transitions by changing its state.

[00164] Typically a FSM has a finite set of states. An extended FSM does not have this limitation and may be used to provide a dynamic DOLSIB. Here the states are not finite, per se; there exists a number of finite state "blocks" on the diagram, but there are also global variables that may store values that take on an infinite number of possibilities. This adds another dimension to the FSM and makes the whole system have an infinite number of "states".

[00165] Figure 25 conceptually illustrates a counter implemented with an EFSM. The counter can count to any desired number using a single idle or initial state. The counter simply outputs the count value and increments this value. Such a counter may be expressed in a DOLSIB language that will be further discussed elsewhere in the application. For example the counter may be represented by the following code:

```
state Idle;
event count;
var value=0;
var str="";
diagram Counter;
Counter (Idle) {
    Idle : count ? value++ -> Idle;
}
```

The code describes the counter beginning in the idle state. Given the count event, the counter will increment the variable value and output this as an action. The arrow ".fwdarw." signifies that idle is the new state. Such code provides a simple way for a programmer to describe the events and actions of the machine.

[00166] According to one embodiment, the Enterprise Interface State Machine (EISM) DOLSIB language may be a C-style DOLSIB EFSM language similar to ISO's Estelle. DOLSIB EISM is based on C style conventions, while Estelle is a Pascal language extension. The ISO Estelle language is defined as a superset of the Pascal language and fits nicely with the language. The DOLSIB EISM language is similar in that it conforms to the syntax of C. The DOLSIB EISM language may provide for more than one state machine diagram (and thus, more than one state machine) to be described and operated by the same script. The state machine may be interpreted allowing for bytecode compilation into a stack machine opcode. The stack-based machine allows for a simple implementation and compilation of the parsed state machines. This also allows for other languages in the future. For

example, a language other than C may be interfaced. The programming language includes a simple ASCII English language equivalent of the C-style convention.

[00167] Figure 26 conceptually illustrates a scheme, files and programs for a state machine 2600 to create an intermediate bytecode that is interpreted by the stack machine, according to one embodiment. An ASCII input file 2610 is written in a DOLSIB EISM language and passed off to a parser 2620. The parser 2620 converts the input file 2610 into byte code 2630, 2640. The byte code 2630 can then be used to run the stack machine 2650 as a state machine or the byte code 2640 may input a dump program 2660 that creates an ASCII output 2670 object file dump of the instructions and the symbol table.

[00168] The parser 2620 may take the input ASCII 2610, parse it for syntax and syntactical errors, and then create a stack machine instruction set 2630 for the resulting state machine 2650. The conversion of the state machine diagram into a stack machine saves time for the run time interpreter and does the preprocessing of the symbol table.

[00169] The interpreter 2650 may be a simple stack machine that receives input events and sends out actions. The stack machine 2650 contains a very limited set of instructions to perform the basic arithmetic and conditional chores at run time.

[00170] The dump program 2660 is a debugging tool. The program 2660 may prints out the symbol table and the instructions for each stack op code.

[00171] Symbols may be used. The symbols may have names that include an alphanumeric string ([A-Za-z_][A-Za-z_{0-9}]*) and the name typically should not that match that of a keyword. Depending on the implementation the names may be case sensitive. Symbols may be declared before they are used and before the first diagram body. Symbols of the same type may be declared within the same command. Symbols may be scalar or vector and the length of a vector may be declared with the symbol type. For example, valid declarations may be: (a) state xyz[4]; (b) event test; (c) var x=0,y=1,z=4; (d) var a="apple",b="banana". There may be different types of symbols. For example, there may be the following five types: diagram, action, event, state, var. Symbols may have a value when evaluated and have a particular function

used as an input to the DOLSIB EISM. Table 1 shows evaluations based on type of symbol.

TABLE 1

Symbol Type	Evaluation
diagram	current state id
action	value of the action; default is zero
event	value of the event if current event; zero otherwise
state	non-zero if this is the current state.
var	an integer or string variable

Table 2 shows assignment actions of symbols of different types when entered into the DOLSIB EISM. The action may be different from the action of the symbols within a program.

TABLE 2

Symbol Type	Assignment Action
diagram	changes the current state of this diagram to the assigned state
action	sets the action value
event	sets the event value
state	ignored
var	sets the variable

The above assignments within the DOLSIB EISM program are valid with the possible exception of state. The state may be a read-only condition of the current system. In which case the programmer may change the state of the diagram within DOLSIB EISM using the diagram assignment.

[00172] Each declared symbol may have an associated integer ID. The integer ID may begin with zero (for the variable "nil"). This variable is usually declared and may be used as an event to trigger transitions. Other symbols may be assigned

beginning with one and incremented according to the order they are declared in the program. An integer may be assigned to each element of a vector. Table 3 illustrates IDs for the previous example:

TABLE 3

Name	Type ID
Nil	int 0
xyz[0]	state 1
xyz[1]	state 2
xyz[2]	state 3
xyz[3]	state 4
test	event 5
x	var 6
y	var 7
z	var 8
a	var 9
b	var 10

[00173] An exemplary program is presented below to further illustrate possible statements in a DOLSIB EISM related program. The symbols in the program may be declared as one of the five types followed by at least one diagram. Consider the simple state diagram that counts the number of incoming count events and dumps the count upon receiving the event dump. This state diagram could be written in DOLSIB EISM as:

```
state Idle;
event count,dump;
var n=0;
diagram Counter;
Counter (Idle) {
Idle : count ? n++ -> Idle
```

| dump ? n(), n=0 -> Idle;

}

All the symbols are declared. The state diagram is named "Counter", and the line Counter (Idle) { begins the diagram definition. The state Idle is placed in parentheses to show that it is the initial state. In this case if no state is declared as the default, the first state in the state diagram is considered the default.

[00174] The state transition may be described in many different formats. The one shown above would have the following meaning: If in state Idle and event count is seen, then increment n and go back to state Idle else if event dump is seen, then output n and n to zero, then go back to state Idle. This state transition may also be written in DOLSIB EISM as:

if Idle and count then n++ enter Idle

else dump then no, n=( ), m=0 enter Idle;

This form may be more understandable. In either case, the keyword "if" is optional. Table 4 shows keywords and symbols that are interchangeable, according to one embodiment.

TABLE 4

Symbol	Meaning
: with and	introduces first arc
? then	follows arc conditional expression
->begin enter	signifies which state to enter if conditional is true
|else elsewith	introduces next arc

A variation on the command structure is the ability to specify outputs without the parentheses. The normal meaning of n would be to output an action that has an ID of n and a value equal to n. For example, one could specify n(5) to set the value of n to

five and then output n. One may also explicitly output a symbol as: Idle with dump then enter Idle output n; instead of the first arc. Notice, that the two proceeding statements may be interchangeable.

[00175] According to one embodiment, the grammar may be specified similar to BNF form. For example, brackets "[ ]" may surround optional items, a vertical bar may be used to show alternatives, and bold symbols may be actual keywords. Consider the following exemplary form:

program	:decl diagrams
decls	:decl | decls decl
decl	:type defs;
type	:**diagram** | **state** | **event** |**int** | **action**
defs	:def | defs, def
def	:lvalue | lvalue = const
diagrams	:diagram | diagrams diagram
diagram	:diagram_init {lines}
diagram_init	:diagram_symbol (state_symbol)| diagramsymbol
lines	:line | lines line
line	:[**if**] states **with** cmds;
with	:**with** | **and** |:
states	:state_symbol | states, state_symbol
cmds	:cmd | cmds **else** cmd
else	:**elsewith** | **else** | |
cmd	:exprs **then** acts **begin** state_symbol [**output** outs]
then	:**then** | **?**
acts	:act | acts, act
act	:lvalue ([expr])
	| expr
begin	:**begin** | **enter** | ->
outs	:lvalue | outs, lvalue

exprs	:const | state_symbol | **&** symbol | lvalue | asgn
	| (expr)
	| expr cmpop expr
	| expr logop expr
	| expr arthop expr
	| **NOT** expr
	| - expr
cmpop	:**LT** | **LE** | **EQ** | **NE** | **GT** | **GE**
logop	:**AND** | **OR**
arthop	:+| - | * | / | %| ..
asgn	:lvalue ASSIGN expr
	| lvalue ++
	| ++ lvalue
	| lvalue --
	| -- lvalue
const	:".*" | [0-9][0-9]*
lvalue	:symbol | symbol [expr]

Core Business Object or Management Object

[00176] The core business or management object that is used to derive other objects may have embedded in it the FSM to be able to parse the EISM DOLSIB with diagrams. Business and management objects may interface with a back-end channel to communicate with enterprise computer systems. The core business or management object may be remotely accessible and may be integrated with the object router. Further, it may have interfaces to the enterprise computer systems and to the FSM. Figure 27 shows code describing a CoreBusinessObject object router, according to one embodiment.

Example Bank Application (BankAccount)

[00177] A service provider may customize the back end communication channel for their intended application and service offerings. In one case the invention is implemented in a way that allows for these different capabilities and customizable features. Consider a simple bank account class that has a balance query and withdraw and deposit methods to change the account balance:

```
class BankAccount {
    int balance() const;
    void withdraw(int amount);
    void deposit(int amount);
    }
```

Given this object, a programmer may query the account balance from within the object-oriented C++ environment. However, the actual mechanics of talking to the back end may vary from merchant to merchant. This may be handled using an intermediate machine to connect with the back-end to communicate using a name-value pair protocol that is modifiable. Consider two banks B1 and B2. B1 may query the back-end for a balance of an account by sending the account number and then the word "balance:query". More specifically, this may be done as follows: (1) send("account_number"), (2) send(<eid>), (3) send("balance:query"), (4) expect("amount"), (5) expect(amount), and (6) return amount.

[00178] B2 may need confirmation that the account number is set and then send the "balance" query. More specifically: (1) send("account-number"), (2) send(<account number>), (3) expect(status), (4) send("balance"), (5) expect(amount), and (6) return amount. Bank B2 has more operations and may have more error conditions.

[00179] Figure 28 conceptually illustrates an exemplary DOLSIB FSM diagram for balance for bank B1, according to one embodiment. These may be used to configure the BankAccount class. The diagram shows B1 being more complicated due to added error transitions. The state diagram may be viewed as an expect script

for a modem that sends out requests and expects back responses matching a particular string. DOLSIB EISM language corresponding to the diagram may be as follows:

```
/* the following are automatically declared by DOLSIB interface:
        state Idle; // default initial state
    state Expect; // waiting on receive == expect_value
    state Found; // default state after receiving expect_value
    state Error; // default error state
    event receive; // indicates the enterprise interface has data
    event timeout; // indicates the timer has expired
    event method; // indicates a method call has started
    action return; // returns from the method call
    action send; // sends data to the enterprise interface
    action throw; // returns from the method call with a throw
    var max_wait=1000; // the default timer value
    var eid=0; // the enterprise id
    var expected_value=""; // waited value
   */
   diagram Balance;
   Balance(Idle) {
    Error : true ? -> Idle
    Idle
     : method == "balance" ?
        timeout(max_wait),
        send(eid),
        send("balance:query")
        expected_value = "amount"
       -> Expect;
    Expect
     : receive == expected_value ?
```

```
        timeout(max_wait)
      -> Found
    | receive != expected_value ?
        throw("expected" ..expected_value
    .."but received"
          ..receive)
      -> Error
    | timeout ?
        throw("timeout while waiting for" ..expected_value)
      -> Error;
    Found
     : receive ?
        return(receive)
      -> Idle
     | timeout ?
        throw("timeout while waiting value")
      -> Idle;
}
```

[00180] Figure 29 conceptually illustrates a diagram including expect, found, and error states. As discussed, the service provider may interact with and expect something from the enterprise interface. Rather than creating a new state for each such "expect" string, a predefined set of states "Expect", "Found" and "Error" may be used. The state transitions defined are for the "Expect" state. A programmer may provide the arcs for the Error and Found states. The defined arcs of the Expect state may have a program similar to the one shown in the example for bank B1:

```
Expect
   : receive == expected_value ?
      timeout(max_wait)
    -> Found
```

```
| receive != expected_value ?
    throw("expected" .. expected_value
  .. "but received"
     .. receive)
  -> Error
| timeout ?
    throw("timeout while waiting for" .. expected_value)
  -> Error;
```

Using the expect state for more than one string (more than one expected response) may include making use of the "extended" nature of the DOLSIB EFSM. Namely, global variables may be used to store the other dimension of state. This is shown with the second bank example using the variable "step".

[00181] Figure 30 conceptually illustrates an exemplary DOLSIB FSM diagram for balance for bank B2, according to one embodiment. The diagram produces a different "name-value" pair protocol. DOLSIB EISM language corresponding to the diagram may be as follows:

```
diagram Balance;
// two steps:
// 1) wait for status.
// 2) wait for "balance".
var step = 1;
Balance(Idle) {
   Error : true ? step = 1 -> Idle;
   Idle
    : method == "balance"?
      timeout(max_wait),
      send("account_number")
      send(eid),
```

```
        expected_value = 0
      -> Expect;
    Found
     : step == 1 ?
        expected_value = "balance",
        step = 1
      -> Expect
     | step == 2 && receive ?
        return(receive),
        step = 1
      -> Idle
     | timeout?
         throw("timeout while waiting balance"),
      -> Error;
}
```

Accordingly, the two bank examples illustrate how different service providers having different back-ends enterprise interactions may use the same business or management object.

[00182] The bank object class structure has been shown and described elsewhere. However, since this may be derived off of the CoreBusinessObject or Management Object the BankAccount object may need an object router definition. An exemplary definition assuming balance, withdraw and deposit methods is as follows:

```
beginclass BankAccount CoreBusinessObject
begindata
enddata
beginmethod
method const int balance
method  void deposit {int amount}
```

```
method  void withdraw {int amount}
endmethod
endclass
```

Hooks may be added to provide method connections to the FSM for this business object:

```
int BankAccountSkel::balance() const {
    fsm_event("balance","");
    RWCString result = fsm_result();
    return atoi(result);
}
```

The balance method calls the FSM event "balance" which starts the diagram transition from Idle to Expect (see Figure 29). Typically, since results in the FSM are performed using strings, the string return type will be converted to an integer.

```
void BankAccountSkel::withdraw(int amount) {
    char a[20];
    sprintf(a,"%d",amount);
    fsm_event("withdraw",a);
    fsm_result();
}
void BankAccountSkel::deposit(int amount) {
    char a[20];
    sprintf(a,"%d",amount);
    fsm_event("deposit",a);
    fsm_result();
}
```

[00183] These examples show that the method fsm_result( ) is called even when not expecting a result. The reason for this is twofold: (1) the thread will block until result is actually called inside of the FSM, and (2) an error result will throw an exception within this fsm_result.

[00184] Figure 31 conceptually illustrates operation of a system 3100 including a thin client, a hub, and node, according to one embodiment. A thin client 3102 (e.g., a client access device) accesses a hub 3110. The hub 3110 includes a user connection server 3112 that connects with a user name server 3114. The user name server 3114 connects with a specified node, in this case node 3150 corresponding to "Alpine" bank. In particular, the connection may include the hub 3110 communicating qualifying identifiers (e.g., <host>, <port>) to the node 3150.

[00185] The node 3150 includes a name server 3152 that after connection accesses a business object 3156. The business object 3156 includes an EFSM 3158 to access a DOLSIB 3162 via a program 3160. The DOLSIB 3162 allows identification of an object 3164. The program 3160 then is able to determine an appropriate "abc" account skeleton object 3166. Based on the qualifiers an object router server 3168 communicates via an object router layer 3170 with a remote client 3116 that uses a bank stub object 3118 to perform remote method execution of bank methods of the node 3150. In particular, an "abc" account stub 3120 may allow determination of an account balance including using a channel server 3172 to interface with a back office 3174. A service management station 3122 and a merchant management station 3124 may perform Events, Configuration, Accounting, Performance, and Security (ECAPS) processing for the hub and the node, respectively.

[00186] Figure 32 conceptually illustrates architecture 3200, according to one embodiment to provide management services, such as ECAPS services, to hub and a node. A client access device 3205 accesses a hub 3210 via a connection 3215 that may support a name-value pair. As shown, the hub 3210 may include a number of modules including an object router 3215 and an object protocol interface 3220 to perform object routing, and a merchant management agent 3225 and a service

management agent 3230 to respectively correspond with a merchant management station 3240 and a service management station 3260.

[00187] A node 3270 may also comprise a number of components as shown, and as discussed elsewhere in the detailed discussion, including an object protocol interface 3275 to assist with object routing, and a merchant management agent 3280 and a service management agent 3285 to respectively interface with the merchant management station 3240 and the service management station 3260. Another node 3290 may be similarly connected with the stations 3240 and 3260 to support management that is desired for the intended application (e.g., ECAPS).

Exemplary Computer Architecture

[00188] As discussed herein, a "system" or "computer system", such as certain client access devices and a system to control a transaction involving multiple service providers, may be an apparatus including hardware and/or software for processing data. The system may include, but is not limited to, a computer (e.g., portable, laptop, desktop, server, mainframe, etc.), hard copy equipment (e.g., optical disk burner, printer, plotter, fax machine, etc.), and the like.

[00189] A computer system 3300 representing an exemplary workstation, host, or server in which features of the present invention may be implemented will now be described with reference to Figure 33. The computer system 3300 represents one possible computer system for implementing embodiments of the present invention, however other computer systems and variations of the computer system 3300 are also possible. The computer system 3300 comprises a bus or other communication means 3301 for communicating information, and a processing means such as processor 3302 coupled with the bus 3301 for processing information. The computer system 3300 further comprises a random access memory (RAM) or other dynamic storage device 3304 (referred to as main memory), coupled to the bus 3301 for storing information and instructions to be executed by the processor 3302. The main memory 3304 also may be used for storing temporary variables or other intermediate information during execution of instructions by the processor 3302. In one embodiment, the main memory 3304 may

be used for storing the operating system, software objects, data structures, coded instructions, rule sets, and other types of data. The computer system 3300 also comprises a read only memory (ROM) and other static storage devices 3306 coupled to the bus 3301 for storing static information and instructions for the processor 3302, such as the BIOS. A data storage device 3307 such as a magnetic disk, zip, or optical disc and its corresponding drive may also be coupled to the computer system 3300 for storing information and instructions.

[00190] The computer system 3300 may also be coupled via the bus 3301 to a display device 3321, such as a cathode ray tube (CRT) or Liquid Crystal Display (LCD), for displaying information to an end user. Typically, a data input device 3322, such as a keyboard or other alphanumeric input device including alphanumeric and other keys, may be coupled to the bus 3301 for communicating information and command selections to the processor 3302. Another type of user input device is a cursor control device 3323, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to the processor 3302 and for controlling cursor movement on the display 3321.

[00191] A communication device 3325 is also coupled to the bus 3301. Depending upon the particular implementation, the communication device 3325 may include a modem, a network interface card, or other well-known interface devices, such as those used for coupling to Ethernet, token ring, or other types of physical attachment for purposes of providing a communication link to support a local or wide area network, for example. In any event, in this manner, the computer system 3300 may be coupled to a number of clients or servers via a conventional network infrastructure, such as a company's intranet, an extranet, or the Internet, for example.

[00192] Embodiments of the invention are not limited to any particular computer system or environment. Rather, embodiments may be used on any stand alone, distributed, networked, or other type of computer system. For example, embodiments may be used on one or more computers compatible with NT, Linux, Windows, Windows NT, Macintosh, any variation of Unix, or others. Embodiments may support ActiveX Controls, Java, web browsers such as Internet Explorer, and standard Web server suites

such as Netscapes' SuiteSpot, FastTrack, Microsoft's Normandy, Microsoft's Commercial Internet System, and others.

[00193] The present invention includes various operations, as described above. The operations of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the operations. The present invention may be provided as a computer-program product that may include a machine-readable medium having stored thereon instructions that may be used to program a computer (or other electronic devices) to perform a process according to the present invention. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, magnet or optical cards, flash memory, or other type of media or machine-readable medium suitable for storing electronic instructions. Moreover, the present invention may also be downloaded as a computer program product, wherein the program may be transferred from a remote computer to a requesting computer by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection). Alternatively, the operations may be performed by a combination of hardware and software.

[00194] In conclusion, the present invention provides an approach for controlling a network transaction involving multiple service providers.

[00195] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.
